

Mendler-style Iso-(Co)inductive predicates: a strongly normalizing approach *

Favio Ezequiel Miranda-Perea
Lourdes del Carmen González-Huesca

Departamento de Matemáticas, Facultad de Ciencias UNAM
Circuito Exterior S/N, Cd. Universitaria, 04510, México D.F., México

favio@ciencias.unam.mx

luglzhuesca@ciencias.unam.mx

We present an extension of the second-order logic AF2 with iso-style inductive and coinductive definitions specifically designed to extract programs from proofs à la Krivine-Parigot by means of primitive (co)recursion principles. Our logic includes primitive constructors of least and greatest fixed points of predicate transformers, but contrary to the common approach, we do not restrict ourselves to positive operators to ensure monotonicity, instead we use the Mendler-style, motivated here by the concept of monotonization of an arbitrary operator on a complete lattice. We prove an adequacy theorem with respect to a realizability semantics based on SAT (saturated) sets and SAT-valued functions and as a consequence we obtain the strong normalization property for the proof-term reduction, an important feature which is absent in previous related work.

Keywords: Mendler-style, (co)inductive definitions, primitive (co)recursion, strong normalization, saturated set, monotonization, second-order logic, programming with proofs.

1 Introduction

The system AF2 for second-order intuitionistic logic introduced by Leivant and Krivine [5, 4], is one of the most fruitful systems obtained by the Curry-Howard correspondence. It types exactly the same terms as the system F of Girard and Reynolds and shares with it the properties of strong normalization and subject reduction. Its main improvement with respect to system F is that it allows the extraction of programs via the programming-with-proofs paradigm of Krivine and Parigot. This method, originally developed in [3] (see also [5]) ensures the correctness of programs (λ -terms) extracted from proofs of termination statements of functions involving formal data types, that is, from proofs of totality. Well known results ensure the extraction of programs for all functions whose termination is provable in second order Peano arithmetic. Nevertheless this result, satisfactory from the extensional point of view does not suffice for an intensional view concerning programs. In AF2 we can get programs for all needed functions, but these do not have necessarily the intended behavior, see [13]. To solve this problem some extensions of AF2 with least fixed points (TTR [14]) and also with greatest fixed points (AF2 ^{μ} [15]) have been introduced. These features allow for the (co)inductive definition of predicates and are suitable for programming with proofs. However the strong normalization is lost due to the use of a fixed-point combinator in the proof-term system, which encodes derivations with lambda terms. The situation is that an iterative function f can be defined within AF2 and therefore its extracted program \tilde{f} is automatically terminating, but the extracted program for a primitive recursive function employs a fixed-point combinator in the extensions of AF2 and therefore its termination is not obvious at all.

*This research is being supported by PAPIIT-UNAM projects IN117711 and IN108810.

This has lead to sophisticated methods to verify that these programs indeed terminate [6], even when they fit into a well-known terminating recursion pattern captured in Gödel's T for the case of natural numbers and generalized to all (co)inductive types in [9, 7, 8], for example. The main contribution of this paper is the introduction of a new extension of AF2 with primitive (co)recursion over least and greatest fixed points, called $\text{AF2}^{M\mu\nu}$, that enjoys the strong normalization property. Instead of using a fixed-point combinator we use the Mendler-style approach of [8] but with two important differences: we use a natural deduction approach, and we do not restrict ourselves to positive operators. This shows that such syntactical restriction is irrelevant to the strong normalization proof of the whole Mendler-system, a feature first discovered by Matthes ([7], p.83) for the inductive fragment. Another contribution of our work is the use of the iso-style, meaning that a (co)inductive predicate and its folding/unfolding are not considered equal but isomorphic. It is important to mention that previous extensions of AF2 with (co)inductive definitions deal only with equi-style predicates, but in our opinion the use of the iso-style is closer to the usual mechanisms of data type definition in functional programming languages. As a consequence of our definition of saturated sets, the proof of the adequacy theorem of our logic does not employ ordinal recursion. Moreover, the rules of our logic are specifically designed to derive statements of totality of functions involving (co)inductive predicates, that is, formulas of the form $\forall x. \mathcal{P}(x) \rightarrow \mathcal{R}(f(x))$. The paper is organized as follows: in section 2 we review the required concepts of fixed-point theory needed to motivate the definition of our logic, which is given in section 3, together with some examples of its expressivity. Section 4 develops the constructions on saturated sets employed in section 5 to define an intuitionistic semantics of the logic. Finally, we discuss related work in section 6 and provide some closing remarks in section 7.

2 Fixed-point theory

In this section we recall some tools of fixed-point theory involving a complete lattice $\langle \mathcal{L}, \sqsubseteq, \sqcap \rangle$, where \sqcap is the infimum operator. Given a monotone operator $\Phi : \mathcal{L} \rightarrow \mathcal{L}$ the Knaster-Tarski theorem guarantees the existence of the least (greatest) fixed-point of Φ , denoted $\text{lfp}(\Phi)$ or $\text{gfp}(\Phi)$, respectively.

Proposition 1 (Conventional (co)induction principles). *Let $\Phi : \mathcal{L} \rightarrow \mathcal{L}$ be a monotone operator on a complete lattice $\langle \mathcal{L}, \sqsubseteq, \sqcap \rangle$. The following holds for every $M \in \mathcal{L}$.*

- *Induction: if $\Phi(M) \sqsubseteq M$ then $\text{lfp}(\Phi) \sqsubseteq M$.*
- *Extended induction: if $\Phi(\text{lfp}(\Phi) \sqcap M) \sqsubseteq M$ then $\text{lfp}(\Phi) \sqsubseteq M$.*
- *Coinduction: if $M \sqsubseteq \Phi(M)$ then $M \sqsubseteq \text{gfp}(\Phi)$.*
- *Extended coinduction¹: if $M \sqsubseteq \Phi(\text{gfp}(\Phi) \sqcup M)$ then $M \sqsubseteq \text{gfp}(\Phi)$.*

Proof. Straightforward. □

The following concepts of monotonization of an arbitrary operator are taken from [7].

Definition 1. *Given an arbitrary operator $\Phi : \mathcal{L} \rightarrow \mathcal{L}$, we define its upper monotonization $\Phi^\sqsupset : \mathcal{L} \rightarrow \mathcal{L}$ and its lower monotonization $\Phi^\sqsubseteq : \mathcal{L} \rightarrow \mathcal{L}$ as $\Phi^\sqsupset(M) = \sqcup \{\Phi(X) \mid X \sqsubseteq M\}$ and $\Phi^\sqsubseteq(M) = \sqcap \{\Phi(X) \mid M \sqsubseteq X\}$.*

The properties and relationships between Φ and its monotonizations are given in the following

Proposition 2. *If $\Phi : \mathcal{L} \rightarrow \mathcal{L}$ is an arbitrary operator then Φ^\sqsubseteq and Φ^\sqsupset are monotone. Moreover,*

¹Recall that in a complete lattice the supremum operator \sqcup can be defined from the infimum operator \sqcap .

- For any $M \in \mathcal{L}$, $\Phi^\sqsubseteq(M) \sqsubseteq \Phi(M) \sqsubseteq \Phi^\sqsupset(M)$.
- If Φ is monotone then $\Phi^\sqsubseteq = \Phi = \Phi^\sqsupset$ and if $\Phi^\sqsubseteq = \Phi$ or $\Phi = \Phi^\sqsupset$ then Φ is monotone.
- $\Phi(\text{lfp}(\Phi^\sqsupset)) \sqsubseteq \text{lfp}(\Phi^\sqsupset)$ and $\text{gfp}(\Phi^\sqsubseteq) \sqsubseteq \Phi(\text{gfp}(\Phi^\sqsubseteq))$.

Proof. Straightforward. \square

Next we justify the Mendler-style (co)induction principles by means of the monotonicizations. This justification is not present in the original work of Mendler ([8]). However, the inductive part is discussed in [7].

Proposition 3 (Mendler (Co)induction principles). *The following holds for any $\Phi : \mathcal{L} \rightarrow \mathcal{L}$ and $M \in \mathcal{L}$.*

- *Induction:* if $\forall X (X \sqsubseteq M \rightarrow \Phi(X) \sqsubseteq M)$ then $\text{lfp}(\Phi^\sqsupset) \sqsubseteq M$.
- *Extended Induction:* if $\forall X (X \sqsubseteq \text{lfp}(\Phi^\sqsupset) \rightarrow X \sqsubseteq M \rightarrow \Phi(X) \sqsubseteq M)$ then $\text{lfp}(\Phi^\sqsupset) \sqsubseteq M$.
- *Coinduction:* if $\forall X (M \sqsubseteq X \rightarrow M \sqsubseteq \Phi(X))$ then $M \sqsubseteq \text{gfp}(\Phi^\sqsubseteq)$.
- *Extended Coinduction:* if $\forall X (\text{gfp}(\Phi^\sqsubseteq) \sqsubseteq X \rightarrow M \sqsubseteq X \rightarrow M \sqsubseteq \Phi(X))$ then $M \sqsubseteq \text{gfp}(\Phi^\sqsubseteq)$.

Proof. The conventional (co)induction principles for Φ^\sqsubseteq and Φ^\sqsupset yield the required principles. For details see [11]. \square

3 The Logic $\text{AF2}^{M\mu\nu}$

We present now the logic $\text{AF2}^{M\mu\nu}$, which is an extension of AF2 with Mendler-style (co)inductive definitions.

- *Terms:* the object terms are defined as usual from a signature Σ including function symbols f of a given arity.

$$t ::= x \mid f(t_1, \dots, t_n)$$

- *Predicates:* apart from the usual predicates (second-order variables or predicate symbols of a signature Σ) we have comprehension predicates, inductive predicate $\mu(\Phi)$ and coinductive predicates $\nu(\Phi)$.

$$\mathcal{P} ::= X \mid P \mid \mathcal{F} \mid \mu(\Phi) \mid \nu(\Phi)$$

here \mathcal{F} is a comprehension predicate of the form $\mathcal{F} =_{\text{def}} \lambda \vec{x}. A$, where A is a formula and its arity is the length of the vector of variables \vec{x} , this predicate intends to represent the set $\{\vec{t} \mid A[\vec{x} := \vec{t}]\}$. On the other hand, Φ is an arbitrary predicate transformer, which is a *closed* expression of the form $\Phi =_{\text{def}} \lambda X. \mathcal{P}$, depending on a second-order variable X . Observe that we do not require any syntactic restriction, like positivity, on the occurrences of X in \mathcal{P} .

- *Formulas:* these are defined as usual

$$A, B ::= \mathcal{P}(t_1, \dots, t_n) \mid A \rightarrow B \mid \forall x A \mid \forall X A$$

- *On equations:* term equations are formulas which play an important role in the logic and are defined as usual in second-order logic: the equation $r = s$ stands for the formula $\forall X. X(r) \rightarrow X(s)$.

The judgments of the logic are of the form $\Gamma \vdash_{\mathbb{E}} t : A$ where $\Gamma = \{x_1 : A_1, \dots, x_n : A_n\}$ is a context of formulas annotated by *proof-term* variables, $\mathbb{E} = \{r_1 = s_1, \dots, r_n = s_n\}$ is a context of equations, A is a formula and t is a *proof-term*, which is a lambda term not to be confused with an object term, for even when we use the same meta-variables for both, object and proof-terms, we consider them to be two completely separated syntactic categories. The derivation relation is inductively defined by means of the following inference rules, where $A[x := r]$ ($A[X := \mathcal{P}]$) always denotes capture-avoiding substitution of first-order (second-order) variables by a term (predicate) in the formula A .

◦ *Rules of AF2:*

$$\begin{array}{c}
\frac{}{\Gamma, x : A \vdash x : A} \text{ (Var)} \quad \frac{\Gamma, x : A \vdash r : B}{\Gamma \vdash \lambda x r : A \rightarrow B} (\rightarrow I) \quad \frac{\Gamma \vdash r : A \rightarrow B \quad \Gamma \vdash s : A}{\Gamma \vdash rs : B} (\rightarrow E) \\
\\
\frac{\Gamma \vdash t : A \quad x \notin FV(\Gamma)}{\Gamma \vdash t : \forall x A} (\forall I) \quad \frac{\Gamma \vdash t : \forall x A}{\Gamma \vdash t : A[x := r]} (\forall E) \\
\\
\frac{\Gamma \vdash t : A \quad X \notin FV(\Gamma)}{\Gamma \vdash t : \forall X A} (\forall^2 I) \quad \frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A[X := \mathcal{P}]} (\forall^2 E) \\
\\
\frac{\Gamma \vdash_{\mathbb{E}} t : A[x := r] \quad \mathbb{E} \triangleright r = s}{\Gamma \vdash_{\mathbb{E}} t : A[x := s]} (Eq)
\end{array}$$

Here $\mathbb{E} \triangleright r = s$ means a derivation of $r = s$ from the set of equations \mathbb{E} according to the following rules:

- $\mathbb{E} \triangleright r = s$, if $r = s$ is a particular case of an equation in \mathbb{E} . That is an equation of the form $r_1[\vec{x} := \vec{t}] = r_2[\vec{x} := \vec{t}]$ or $r_2[\vec{x} := \vec{t}] = r_1[\vec{x} := \vec{t}]$, where $r_1 = r_2 \in \mathbb{E}$ and \vec{t} are arbitrary terms.
- $r = s$ was obtained from \mathbb{E} by reflexivity, transitivity or compatibility with functions, that is, by one of the following rules:

$$\begin{array}{c}
\frac{}{\mathbb{E} \triangleright r = r} \quad \frac{\mathbb{E} \triangleright r = s \quad \mathbb{E} \triangleright s = t}{\mathbb{E} \triangleright r = t} \quad \frac{\mathbb{E} \triangleright r_1 = s_1 \quad \dots \quad \mathbb{E} \triangleright r_n = s_n}{\mathbb{E} \triangleright f(r_1, \dots, r_n) = f(s_1, \dots, s_n)}
\end{array}$$

- *Rules involving (co)inductive definitions:* these rules are specifically designed to construct (de-struct) elements of an inductive (coinductive) predicate and to prove statements of totality of functions. Given two n -ary² predicates \mathcal{P}, \mathcal{R} , and a vector \vec{g} of n function symbols, the following notation will be used: $\mathcal{P} \subseteq_{\vec{g}} \mathcal{R}$ is the formula $\forall \vec{x}. \mathcal{P}(\vec{x}) \rightarrow \mathcal{R}(\vec{g}(\vec{x}))$, where, in general, a vector application of $\vec{f} =_{\text{def}} f_1, \dots, f_n$ to $\vec{t} =_{\text{def}} t_1, \dots, t_n$, denoted $\vec{f}(\vec{t})$, is defined as $\vec{f}(\vec{t}) =_{\text{def}} f_1(t_1), \dots, f_n(t_n)$. In particular $\mathcal{P} \subseteq \mathcal{R}$ is the formula $\forall \vec{x}. \mathcal{P}(\vec{x}) \rightarrow \mathcal{R}(\vec{x})$ or even $\mathcal{P} \rightarrow \mathcal{R}$, if the predicates have arity 0. Given a predicate transformer $\Phi =_{\text{def}} \lambda X. \mathcal{P}$ and a predicate \mathcal{R} , the application of Φ to \mathcal{R} , is defined by $\Phi(\mathcal{R}) =_{\text{def}} \mathcal{P}[X := \mathcal{R}]$, clearly $\Phi(\mathcal{R})$ is a predicate. The following rules are motivated by the last part of proposition 2 and by proposition 3, for lattices of sets. It is important to observe that in each rule we employ $\mu(\Phi)$ or $\nu(\Phi)$ instead of the expected $\mu(\Phi^{\sqsupset})$ or $\nu(\Phi^{\sqsubseteq})$. This choice will be justified by the semantics.

²We are mostly interested in predicates for data types, which means $n = 1$. However we present the system for any arity for the sake of generality.

- *Inductive construction and coinductive destruction*: for any (co)inductive predicate $\mu(\Phi)$ or $\nu(\Phi)$ of arity n , we assume a fixed set of n function symbols \vec{c} or \vec{d} , called the constructors of $\mu(\Phi)$ or the destructors of $\nu(\Phi)$.

$$\frac{\Gamma \vdash r : \Phi(\mu(\Phi))(\vec{t})}{\Gamma \vdash \text{in } r : \mu(\Phi)(\vec{c}(\vec{t}))} (\mu I) \quad \frac{\Gamma \vdash r : \nu(\Phi)(\vec{t})}{\Gamma \vdash \text{out } r : \Phi(\nu(\Phi))(\vec{d}(\vec{t}))} (\nu E)$$

These rules correspond to the last part of proposition 2, but observe that our (co)inductive predicates are in iso-style, due to the presence of the constructors \vec{c} (destructors \vec{d}). Moreover, the equi-style can be easily recovered by using as constructors/destructors the identity function symbol id while adding $id(x) = x$ to the equational axioms.

- *Primitive recursion*: this rule is modelled after the Mendler extended induction principle given by proposition 3. Here we regard a composition $f \circ c$ as a new function symbol defined by the equation $(f \circ c)(x) = f(c(x))$ and a composition of tuples $\vec{f} \circ \vec{c}$ as the tuple $f_1 \circ c_1, \dots, f_n \circ c_n$.

$$\frac{\Gamma \vdash s : \forall X (X \subseteq \mu(\Phi) \rightarrow X \subseteq_{\vec{f}} \mathcal{K} \rightarrow \Phi(X) \subseteq_{\vec{f} \circ \vec{c}} \mathcal{K}) \quad \Gamma \vdash r : \mu(\Phi)(\vec{t})}{\Gamma \vdash \text{MRec } s \ r : \mathcal{K}(\vec{f}(\vec{t}))} (\mu E)$$

- *Primitive corecursion*: the Mendler extended coinduction principle of proposition 3 inspires the following rule. Observe that in both rules (recursion and corecursion), we can recover the corresponding exact principle of proposition 3 by using the equi-style and by regarding f as the identity function via the equation $f(x) = x$.

$$\frac{\Gamma \vdash s : \forall X (\nu(\Phi) \subseteq X \rightarrow \mathcal{K} \subseteq_{\vec{f}} X \rightarrow \mathcal{K} \subseteq_{\vec{d} \circ \vec{f}} \Phi(X)) \quad \Gamma \vdash r : \mathcal{K}(\vec{t})}{\Gamma \vdash \text{MCoRec } s \ r : \nu(\Phi)(\vec{f}(\vec{t}))} (\nu I)$$

- *Operational semantics*: To end the definition of our logic, we define the operational semantics of the proof-term reduction, which is given by the one-step reduction relation $t \rightarrow_{\beta} t'$ defined as the closure of the following axioms under all term formers.

$$\begin{aligned} (\lambda x r)s &\mapsto_{\beta} r[x := s] \\ \text{MRec } s(\text{in } t) &\mapsto_{\beta} s(\lambda x x)(\text{MRec } s)t \\ \text{out}(\text{MCoRec } s \ t) &\mapsto_{\beta} s(\lambda x x)(\text{MCoRec } s)t \end{aligned}$$

Here and throughout the paper $\text{MRec } s$ means $\lambda x. \text{MRec } s \ x$ and the same is true for $\text{MCoRec } s$.

- *Derived rules*: To simplify the presentation of examples we will employ the usual second-order encodings for conjunctions, disjunctions and existential formulas, which allow to obtain the following derived rules for judgements and operational semantics:

$$\begin{aligned} \frac{\Gamma \vdash r : A \quad \Gamma \vdash s : B}{\Gamma \vdash \langle r, s \rangle : A \wedge B} (\wedge I) \quad & \frac{\Gamma \vdash s : A \wedge B}{\Gamma \vdash \text{fst } s : A} (\wedge E_L) \quad & \frac{\Gamma \vdash s : A \wedge B}{\Gamma \vdash \text{snd } s : B} (\wedge E_R) \\ \frac{\Gamma \vdash r : A}{\Gamma \vdash \text{inl } r : A \vee B} (\vee I_L) \quad & \frac{\Gamma \vdash r : B}{\Gamma \vdash \text{inr } r : A \vee B} (\vee I_R) \\ \frac{\Gamma \vdash r : A \vee B \quad \Gamma, x : A \vdash s : C \quad \Gamma, y : B \vdash t : C}{\Gamma \vdash \text{case}(r, x.s, y.t) : C} (\vee E) \end{aligned}$$

$$\begin{array}{c}
\frac{\Gamma \vdash t : A[x := r]}{\Gamma \vdash \text{pack } t : \exists x.A} \qquad \frac{\Gamma \vdash t : \exists x.A \quad \Gamma, u : A \vdash r : B \quad x \notin FV(\Gamma, B)}{\Gamma \vdash \text{open}(t, u.r) : B} \\
\\
\begin{array}{ccc}
\text{fst}\langle r, s \rangle & \mapsto_{\beta} & r \\
\text{case}(\text{inl } r, x.s, y.t) & \mapsto_{\beta} & s[x := r]
\end{array}
\qquad
\begin{array}{ccc}
\text{snd}\langle r, s \rangle & \mapsto_{\beta} & s \\
\text{case}(\text{inr } r, x.s, y.t) & \mapsto_{\beta} & t[y := r]
\end{array} \\
\\
\text{open}(\text{pack } t, u.r) \mapsto_{\beta} r[u := t]
\end{array}$$

The proof-reduction behaves well with respect to the derivation relation, as ensured by the following

Proposition 4 (Subject-reduction of $\text{AF2}^{M\mu\nu}$). *If $\Gamma \vdash_{\mathbb{E}} t : A$ and $t \rightarrow^* t'$ then $\Gamma \vdash_{\mathbb{E}} t' : A$.*

Proof. The proof is not trivial since $\text{AF2}^{M\mu\nu}$ is formulated in Curry-style and it is analogous to the one developed in [9] for a similar system. \square

3.1 On (Co)Iteration

In fixed-point theory, (co)iteration can be easily derived from primitive (co)recursion. This is not the case for conventional (co)induction principles in type theory like the ones developed in [9] (see section 4.5 of [7] for a deep discussion on this subject) and therefore (co)iterators must be defined apart from (co)recursors. For the Mendler-style, (co)iterators correspond to the (co)induction principles of proposition 3, and are again superfluous (as noticed also in [7]). Let us define $\text{Mlt } s r =_{\text{def}} \text{MRec } s' r$ and $\text{MColt } s r =_{\text{def}} \text{MCoRec } s' r$, where $s' =_{\text{def}} \lambda _ . s$ and $_$ is a dummy variable. The following rules for inference and proof-reduction are derivable:

- Iteration

$$\frac{\Gamma \vdash s : \forall X (X \subseteq_{\vec{f}} \mathcal{K} \rightarrow \Phi X \subseteq_{\vec{f} \circ \vec{c}} \mathcal{K}) \quad \Gamma \vdash r : \mu(\Phi)(\vec{t})}{\Gamma \vdash \text{Mlt } s r : \mathcal{K}(\vec{f}(\vec{t}))} (\mu E^-)$$

- Coiteration

$$\frac{\Gamma \vdash s : \forall X (\mathcal{K} \subseteq_{\vec{f}} X \rightarrow \mathcal{K} \subseteq_{\vec{d} \circ \vec{f}} \Phi X) \quad \Gamma \vdash r : \mathcal{K}(\vec{t})}{\Gamma \vdash \text{MColt } s r : \nu(\Phi)(\vec{f}(\vec{t}))} (\nu I^-)$$

$$\text{Mlt } s(\text{int}) \rightarrow s(\text{Mlt } s) t$$

$$\text{out}(\text{MColt } s t) \rightarrow s(\text{MColt } s) t$$

We will use both the (co)iteration and the primitive (co)recursion rules in the examples that we discuss next.

3.2 Examples

In this section we develop some examples of (co)inductive predicates that show the expressivity of our logic. Due to lack of space a deep discussion about the advantages and disadvantages of both the iso-style and the equi-style is missing. Instead, we provide some examples that show some of such (dis)advantages. Every program (λ -term) \vec{f} presented here is extracted from a proof of totality for a function f involving (co)inductive predicates and specified by a set of equations in the logic. Moreover, the reader can verify that in each case \vec{f} is operationally correct.

Example 1 (Iso-inductive ad-hoc Natural Numbers). *Let $\langle \rangle =_{\text{def}} \lambda x.x = \star$ where \star is a fixed constant, this comprehension predicate is called unit predicate and represents a type with unique inhabitant \star . We define the predicate of natural numbers as $\mathbb{N} =_{\text{def}} \mu(\Phi)$ where $\Phi =_{\text{def}} \lambda X. \lambda x. \langle \rangle(x) \vee X(x)$, taking the*

successor function suc as constructor and $\text{suc}(\star) = 0$ as equational axiom. Defining $\bar{0} =_{\text{def}} \text{in}(\text{inl}())$,³ and $\bar{\text{suc}} =_{\text{def}} \lambda x. \text{in}(\text{inr } x)$ we can show that $\vdash \bar{0} : \mathbb{N}(0)$ and $\vdash \bar{\text{suc}} : \forall x. \mathbb{N}(x) \rightarrow \mathbb{N}(\text{suc } x)$. We call this an ad-hoc definition, for zero is in the image of the successor and therefore our representation is not compatible with Peano's axioms. This is an unpleasant feature which can be avoided at some cost (see example 3). However, operationally, our definition is adequate. For instance, the sum and factorial are programmed as follows:

- *Sum*: from $\mathbb{E}_{\text{sum}} = \{\text{sum } n \ 0 = n, \text{sum } n (\text{suc } m) = \text{suc}(\text{sum } n \ m)\}$, we get $\vdash_{\mathbb{E}_{\text{sum}}} \bar{\text{sum}} : \forall n. \forall x. \mathbb{N}(n) \rightarrow \mathbb{N}(x) \rightarrow \mathbb{N}(\text{sum } n \ x)$, where $\bar{\text{sum}} =_{\text{def}} \lambda n. \text{Mlts}$ and $s =_{\text{def}} \lambda y \lambda z. \text{case}(z, u.n, v.\bar{\text{suc}}(yv))$. This program behaves correctly: $\bar{\text{sum}} \ n \ \bar{0} \rightarrow^* n$ and $\bar{\text{sum}} \ n (\bar{\text{suc}} \ m) \rightarrow^* \bar{\text{suc}}(\bar{\text{sum}} \ n \ m)$.
- *Factorial*: using the equations $\mathbb{E}_{\text{fac}} = \{\text{fac } 0 = 1, \text{fac}(\text{suc } n) = (\text{suc } n) * (\text{fac } n)\}$, we can derive $\vdash_{\mathbb{E}_{\text{fac}}} \bar{\text{fac}} : \forall x. \mathbb{N}(x) \rightarrow \mathbb{N}(\text{fac } x)$, where $\bar{\text{fac}} =_{\text{def}} \text{MRec } s$ and the step term s is defined as $s =_{\text{def}} \lambda y \lambda z \lambda w. \text{case}(w, u.\bar{1}, v.\bar{\text{suc}}(yv) * (zv))$.

The reader should convince herself that the naive definition of natural numbers coming from fixed point theory, given by the predicate transformer $\Phi =_{\text{def}} \lambda X. \lambda x. x = 0 \vee X(x)$, does not work. In the equi-inductive approach we cannot construct any number other than zero, and in the iso-inductive case we cannot construct the zero. Another possibility is the one taken in [16], discussed next.

Example 2 (Equi-inductive Natural Numbers). We define $\mathbb{N} =_{\text{def}} \mu(\Phi)$ with the predicate transformer $\Phi = \lambda X. \lambda x. Z(x) \vee X(p(x))$ where $Z =_{\text{def}} \lambda x. x = 0$ and p is a function symbol, whose intended meaning is the predecessor function. We have $\vdash \bar{0} : \mathbb{N}(0)$ and $\vdash \bar{p} : \forall x. \mathbb{N}(p(x)) \rightarrow \mathbb{N}(x)$ where $\bar{0} =_{\text{def}} \text{in}(\text{inl}())$ and $\bar{p} =_{\text{def}} \lambda x. \text{in}(\text{inr } x)$. In this case we have the following derivation: $f_0 : \forall x. Z(x) \rightarrow \text{It}(x)$, $f_p : \forall x. \text{It}(p(x)) \rightarrow \text{It}(x) \vdash \text{Mlts} : \forall x. \mathbb{N}(x) \rightarrow \text{It}(x)$, where $s =_{\text{def}} \lambda x. \lambda y. \text{case}(y, u.f_0(u), v.f_p(xv))$ and $\text{It}(x)$ is a predicate representing the fact that the image of a given function f on x was defined by iteration. If we set $g =_{\text{def}} \text{Mlts}$ then $g \bar{0} \rightarrow^* f_0()$ and $g(\bar{p} \ n) \rightarrow^* f_p(g \ n)$. This example shows that our logic subsumes the Mendler-style programming methodology of [16]. However, this approach does not correspond to the idea of programming with proofs that we pursue.

Our final version of natural numbers shows the full use of the iso-inductive style and depends on the disjoint union of predicates \uplus which is a predicate that can be defined under the presence of the Parigot's restriction operator \upharpoonright (see [14]). This operator can be added to our logic without a problem and behaves as a conjunction where the right formula is an equation without algorithmic content.⁴ Defining $\mathcal{P} \uplus \mathcal{R} =_{\text{def}} \lambda x. \exists z. (\mathcal{P}(z) \upharpoonright x = \text{lf } z) \vee (\mathcal{R}(z) \upharpoonright x = \text{rg } z)$ we get that $\Gamma \vdash r : \mathcal{P}(t)$ implies $\Gamma \vdash \text{pack}(\text{inl } r) : (\mathcal{P} \uplus \mathcal{R})(\text{lf } t)$ or $\Gamma \vdash \text{pack}(\text{inr } r) : (\mathcal{R} \uplus \mathcal{P})(\text{rg } t)$. One important advantage of using this predicate together with our iso-style is that we do not need to deal directly with existential formulas in definitions, and therefore the following examples are closer to the data type definition mechanisms of functional programming languages.

Example 3 (Iso-inductive Natural Numbers). The natural numbers are given now by the inductive definition $\mathbb{N} =_{\text{def}} \mu(\Phi)$ where $\Phi = \lambda X. \langle \rangle \uplus X$, and we use a generic constructor cnat , which yields the usual constructors by adopting the equational axioms $0 = \text{cnat}(\text{lf } \star)$ and $\text{suc } x = \text{cnat}(\text{rg } x)$. These constructors are implemented by $\bar{0} =_{\text{def}} \text{in}(\text{pack}(\text{inl}()))$ and $\bar{\text{suc}} =_{\text{def}} \lambda z. \text{in}(\text{pack}(\text{inr } z))$. Let us present the extracted programs for sum, factorial and predecessor:

- *Sum*: from $\mathbb{E}_{\text{sum}} = \{\text{sum } n \ 0 = n, \text{sum } n (\text{suc } m) = \text{suc}(\text{sum } n \ m)\}$, we derive $\vdash_{\mathbb{E}_{\text{sum}}} \lambda n. \text{Mlts} : \forall n. \forall x. \mathbb{N}(n) \rightarrow \mathbb{N}(x) \rightarrow \mathbb{N}(\text{sum } n \ x)$ where $s =_{\text{def}} \lambda y. \lambda z. \text{open}(z, u. \text{case}(u, v.n, w.\bar{\text{suc}}(yw)))$. Therefore we get $\bar{\text{sum}} =_{\text{def}} \lambda n. \text{Mlts}$.

³Sometimes an equation is involved directly in a judgment and we agree to give it the void proof-term $()$ as code.

⁴That is, an equation that is not codified by a proof-term.

- *Factorial*: from $\mathbb{E}_{\text{fac}} = \{\text{fac}(0) = 1, \text{fac}(\text{suc}(n)) = \text{suc}(n) * \text{fac}(n)\}$, we derive $\vdash_{\mathbb{E}_{\text{fac}}} \text{MRec } s : \forall x. \mathbb{N}(x) \rightarrow \mathbb{N}(\text{fac } x)$ where $s =_{\text{def}} \lambda y. \lambda z. \lambda w. \text{open}(w, u. \text{case}(u, u_1. \bar{1}, u_2. \overline{\text{suc}}(yu_2) \bar{\star}(zu_2)))$. Therefore $\overline{\text{fac}} =_{\text{def}} \text{MRec } s$ is a correct program for the factorial.
- *Predecessor*: an efficient handling-error predecessor specified by $\mathbb{E}_{\text{pred}} = \{\text{error} = \text{If } \star, \text{pred } 0 = \text{error}, \text{pred}(\text{suc } n) = \text{rgn}\}$, is implemented by $\overline{\text{pred}} =_{\text{def}} \text{MRec } s$, where the step function is $s =_{\text{def}} \lambda y. \lambda z. \lambda w. \text{open}(w, u. \text{case}(u, u_1. \text{pack}(\text{inl}()), u_2. \text{pack}(\text{inr}(yu_2)))$, for we derive $\vdash_{\mathbb{E}_{\text{pred}}} \text{MRec } s : \forall x. \mathbb{N}(x) \rightarrow (\langle \rangle \uplus \mathbb{N})(\text{pred } x)$.

In a similar way to the last example, we can define all usual inductive data types like finite lists or trees (see [11, 9] for several related examples). We present next, coinductive predicates corresponding to the conatural numbers and the lazy data type of streams or strictly infinite lists. These examples show that we can deal with infinite objects within a terminating system. It is important to observe that in the former case the iso-style is more convenient, and for the latter the equi-style suffices.

The implementation of the predicate for the so-called conatural numbers, corresponding to the ordinal $\omega + 1$, gives us the opportunity to show the use of corecursion to construct inhabitants of data types with infinite objects, in this case the ordinal ω . We observe that the implementation of conatural numbers, as well as the implementations for natural numbers discussed above, do not correspond to Church numerals, as it happens in AF2. In particular the normal proof-term coding the fact that $\text{CoNat}(\omega)$ holds does not involve an “infinite” Church numeral, which would be a non-terminating term, for ω is specified as a conatural number that equals its predecessor and will be constructed by means of corecursion.

Example 4 (Iso-coinductive conatural numbers). *The conatural numbers are defined by $\text{CoNat} =_{\text{def}} \nu(\Phi)$, where $\Phi =_{\text{def}} \lambda X. \lambda x. \langle \rangle(x) \vee X(x)$, and taking the predecessor function pred as destructor with implementation $\overline{\text{pred}} =_{\text{def}} \text{out}$. Let us construct the conatural numbers by means of corecursion.*

- *Zero*: let 0 be a constant, z be a unary function symbol and $\mathbb{E}_z = \{\text{pred}(z(x)) = \star, 0 = z(\star)\}$. If we define $\bar{0} =_{\text{def}} \text{MCoRec } s()$, where $s =_{\text{def}} \lambda x \lambda y. \lambda u. \text{inl } u$, then $\vdash_{\mathbb{E}_z} \bar{0} : \text{CoNat}(0)$ and $\overline{\text{pred}} \bar{0} \rightarrow^* \text{inl}()$.
- *Succesor*: let suc be a unary function and $\mathbb{E}_{\text{suc}} = \{\text{pred}(\text{suc } x) = x\}$. We have $\vdash \overline{\text{suc}} : \forall x. \text{CoNat}(x) \rightarrow \text{CoNat}(\text{suc}(x))$, where $\overline{\text{suc}} =_{\text{def}} \text{MCoRec } s$ and $s =_{\text{def}} \lambda x \lambda y. \lambda z. \text{inr}(xz)$. Moreover, the operational semantics yields $\overline{\text{pred}}(\overline{\text{suc}} n) \rightarrow^* \text{inr } n$.
- *Omega*: to define the infinite ordinal ω , we use a unary function ω^\dagger and axioms $\mathbb{E}_{\omega^\dagger} = \{\omega = \omega^\dagger(\star), \text{pred}(\omega^\dagger(x)) = \omega^\dagger(x)\}$. Then we get $\vdash \overline{\omega^\dagger} : \forall x. \langle \rangle(x) \rightarrow \text{CoNat}(\omega^\dagger(x))$. By defining $\overline{\omega} =_{\text{def}} \overline{\omega^\dagger}()$ we get $\vdash \overline{\omega} : \text{CoNat}(\omega)$. The needed proof-term is given by $\overline{\omega^\dagger} =_{\text{def}} \text{MCoRec } s$, where $s =_{\text{def}} \lambda x \lambda y \lambda z. \text{inr}(yz)$.

Our last example of a coinductive predicate corresponds to streams or strictly infinite lists.

Example 5 (Equi-coinductive Streams). *The streams over a data type A are defined as $\mathbb{S}_A =_{\text{def}} \nu(\Phi)$ where $\Phi =_{\text{def}} \lambda X. \lambda x. A(\text{head}(x)) \wedge X(\text{tail}(x))$, and the destructor d is the identity function. The programs for the usual destructors are $\overline{\text{head}} =_{\text{def}} \lambda x. \text{fst}(\text{out } x)$ and $\overline{\text{tail}} =_{\text{def}} \lambda x. \text{snd}(\text{out } x)$, extracted from $\vdash \overline{\text{head}} : \forall x. \mathbb{S}_A(x) \rightarrow A(\text{head } x)$ and $\vdash \overline{\text{tail}} : \forall x. \mathbb{S}_A(x) \rightarrow \mathbb{S}_A(\text{tail } x)$. We present now some programs involving streams:*

- *The function from, that generates the stream of natural numbers from a given one, is specified by $\mathbb{E}_{\text{from}} = \{\text{head}(\text{from } x) = x, \text{tail}(\text{from } x) = \text{from}(\text{suc } x)\}$. The reader can verify that $\vdash_{\mathbb{E}_{\text{from}}} \text{from} : \forall x. \mathbb{N}(x) \rightarrow \mathbb{S}_{\mathbb{N}}(\text{from } x)$ where $\overline{\text{from}} =_{\text{def}} \text{MColts}$ and $s =_{\text{def}} \lambda y \lambda z. \langle z, y(\overline{\text{suc}} z) \rangle$, and that $\overline{\text{head}}(\text{from } x) \rightarrow^* x$ and $\overline{\text{tail}}(\text{from } x) \rightarrow^* \text{from}(\overline{\text{suc}} x)$.*

- The constructor `cons` is defined by $\mathbb{E}_{\text{cons}} = \{\text{head}(\text{cons } xy) = x, \text{tail}(\text{cons } xy) = y\}$ and requires corecursion to be implemented. We get a program $\overline{\text{cons}}$ from the proof $\vdash_{\mathbb{E}_{\text{cons}}} \overline{\text{cons}} : \forall x \forall y. A(x) \rightarrow \mathbb{S}_A(y) \rightarrow \mathbb{S}_A(\text{cons } xy)$ where $\overline{\text{cons}} =_{\text{def}} \lambda x. \text{MCoRec } s$ and $s =_{\text{def}} \lambda f_1 \lambda f_2 \lambda w. \langle x, f_1 w \rangle$.
- The function `map` on streams is specified by $\mathbb{E}_{\text{map}} = \{\text{head}(\text{map } f \ell) = f(\text{head } \ell), \text{tail}(\text{map } f \ell) = \text{map } f(\text{tail } \ell)\}$. An extracted program from $\vdash_{\mathbb{E}_{\text{map}}} \overline{\text{map}} : (\forall x. A(x) \rightarrow B(f(x))) \rightarrow \forall z. \mathbb{S}_A(z) \rightarrow \mathbb{S}_B(\text{map } f z)$ is $\overline{\text{map}} =_{\text{def}} \lambda f. \text{MColt } s$, where $s =_{\text{def}} \lambda y \lambda z. \langle f(\text{head } x), y(\text{tail } x) \rangle$.
- A function similar to `map` but that requires corecursion in the implementation is `maphd`, which applies a given function only to the head of a stream. It is defined by $\mathbb{E}_{\text{maphd}} = \{\text{head}(\text{maphd } f \ell) = f(\text{head } \ell), \text{tail}(\text{maphd } f \ell) = \text{tail } \ell\}$. We get the program $\vdash_{\mathbb{E}_{\text{maphd}}} \overline{\text{maphd}} : (\forall x. A(x) \rightarrow A(f(x))) \rightarrow \forall z. \mathbb{S}_A(z) \rightarrow \mathbb{S}_A(\text{maphd } f z)$ where $\overline{\text{maphd}} =_{\text{def}} \lambda f. \text{MCoRec } s$ and the step function s is defined by $\lambda y. \lambda z. \lambda w. \langle f(\text{head } x), \text{tail } x \rangle$.

We finish the section with a couple of examples involving binary predicates.

Example 6 (Iso-inductive order in natural numbers). *The following recursive definition of order for natural numbers:*

$$\frac{\mathbb{N}(n)}{0 < \text{suc } n} \qquad \frac{n < m}{\text{suc } n < \text{suc } m}$$

is implemented by the iso-inductive definition $\mathbb{L} = \mu(\Phi)$ where the predicate transformer is $\Phi =_{\text{def}} \lambda X^{(2)}. \lambda x, y. (x = 0 \wedge \mathbb{N}(y)) \vee \exists z. X(z, y) \upharpoonright (x = \text{suc } z)$, and the constructors are the identity and the successor functions $\vec{c} =_{\text{def}} \text{Id}, \text{suc}$. The derivations $\vdash \lambda n. \text{in}(\text{inl}(\langle \rangle, n)) : \forall n. \mathbb{N}(n) \rightarrow \mathbb{L}(0, \text{suc } n)$ and $\vdash \lambda w. \text{in}(\text{inr}(\text{pack } w)) : \forall n \forall m. \mathbb{L}(n, m) \rightarrow \mathbb{L}(\text{suc } n, \text{suc } m)$ can be easily verified.

Example 7 (Equi-coinductive observational equality for streams). *Leibniz equality is not always adequate for reasoning about streams (see [15]), in some cases it is better to employ the observational equality. This equality relation is defined by the equi-coinductive binary predicate $\mathcal{E} =_{\text{def}} \nu(\Phi)$ where $\Phi =_{\text{def}} \lambda X^{(2)}. \lambda x, y. \text{head } x = \text{head } y \wedge X(\text{tail } x, \text{tail } y)$. It is immediate to verify that $\vdash \lambda x. \text{fst}(\text{out } x) : \forall x \forall y. \mathcal{E}(x, y) \rightarrow \text{head } x = \text{head } y$ and $\vdash \lambda x. \text{snd}(\text{out } x) : \forall x. \forall y. \mathcal{E}(x, y) \rightarrow \mathcal{E}(\text{tail } x, \text{tail } y)$. Moreover, the corecursion rule yields $\vdash e : \forall x \forall y. \text{head } x = \text{head } y \rightarrow \mathcal{E}(\text{tail } x, \text{tail } y) \rightarrow \mathcal{E}(x, y)$, where the proof term e is given by $e =_{\text{def}} \lambda x \lambda y. \text{MCoRec } s \langle x, y \rangle$ and $s =_{\text{def}} \lambda w. \lambda u. \lambda v. \langle \text{fst } v, w(\text{snd } v) \rangle$. These proofs imply that two streams are observationally equal if and only if their heads are equal and their tails are again observationally equal.*

4 Saturated Sets

We develop here all constructions on a complete lattice of so-called saturated sets needed to define the semantics of the logic. It is important to emphasize that in this section a term is exclusively a λ -term belonging to the set $\Lambda = \{t \mid t \text{ is a proof-term of } \text{AF2}^{M\mu\nu}\}$.

Definition 2. A term t is called an *I-term* if it was generated by an introduction rule, i.e., *I-terms* are terms of the following shapes: $\lambda x r$, $\text{in } r$, $\text{MCoRec } s r$. Analogously *E-terms* are terms generated by an elimination rule, i.e. they are terms of the following shapes: $r s$, $\text{out } r$, $\text{MRec } s r$.

Observe that any term is either a variable, an *I-term* or an *E-term*.

Instead of reasoning with infinite reduction sequences we will work with an inductive definition of a set SN including all strongly normalizing terms. We discuss its definition now.

Definition 3. *Evaluation contexts are defined by the following grammar:*

$$E[\bullet] ::= \bullet \mid E[\bullet]s \mid \text{out } E[\bullet] \mid \text{MRec } s E[\bullet]$$

Let us observe that an evaluation context may be considered as an E -term with a unique placeholder \bullet . Therefore, evaluation contexts are sometimes called elimination contexts or multiple eliminations. In the following, we will write $E[r]$ for the E -term obtained by substituting the placeholder \bullet by the term r in $E[\bullet]$. That is $E[r] =_{\text{def}} E[\bullet][\bullet := r]$ where the substitution is defined as if \bullet were a term variable. A term of the form $E[x]$ is called a *neutral term*. The notion of *weak head reduction*, denoted \rightarrow_{whd} , needed to define the set SN is defined as follows:

$$\frac{t \rightarrow_{\beta} t'}{E[t] \rightarrow_{\text{whd}} E[t']}$$

The final concept involved in the inductive definition of the set SN is the set $\text{ist}(t)$ of immediate subterms of a given term t , defined as follows: $\text{ist}(x) = \emptyset$, $\text{ist}(\lambda x r) = \text{ist}(\text{in } r) = \text{ist}(\text{out } r) = \{r\}$, $\text{ist}(rs) = \text{ist}(\text{MRec } sr) = \text{ist}(\text{MCoRec } sr) = \{s, r\}$. We will also need the set $\text{ist}(E[\bullet])$ of immediate subterms of a given evaluation context which is defined as if $E[\bullet]$ were a term.

Definition 4. *The set SN is defined by means of the following inductive definition:*

$$\begin{array}{c} \frac{}{x \in \text{SN}} \quad (\text{SN-VAR}) \qquad \frac{t \text{ is an I-term} \quad \text{ist}(t) \subseteq \text{SN}}{t \in \text{SN}} \quad (\text{SN-I}) \\[10pt] \frac{E[x] \in \text{SN} \quad \text{ist}(E[\bullet]) \subseteq \text{SN}}{E'[E[x]] \in \text{SN}} \quad (\text{SN-E}) \qquad \frac{E[t'] \in \text{SN} \quad E[t] \rightarrow_{\text{whd}} E[t'] \quad \text{prt}(t) \subseteq \text{SN}}{E[t] \in \text{SN}} \quad (\text{SN-W}) \end{array}$$

where for a redex t , $\text{prt}(t)$ is the set of problematic subterms of t , which are the terms that might break the strong normalization of t , even knowing that its reduct t' strongly normalizes. This set is defined as follows: $\text{prt}((\lambda x.r)s) = \{s\}$, $\text{prt}(\text{MRec } s(\text{in } r)) = \text{prt}(\text{out}(\text{MCoRec } sr)) = \emptyset$.

It can be proved that the characterization SN of the set of strongly normalizing terms is sound, that is: if $t \in \text{SN}$ then there is no infinite reduction sequence $t \rightarrow t_1 \rightarrow t_2 \rightarrow \dots$.

Now we can define a concept of saturated set, modelled after the definition of SN.

Definition 5 (SAT-set). *A set of terms \mathcal{M} is saturated if and only if it consists only of terms in SN, it contains all neutral terms of SN, and it is closed under weak head expansion of SN terms. This can elegantly be defined by the following rules:*

$$\begin{array}{c} \frac{t \in \mathcal{M}}{t \in \text{SN}} \quad (\text{SAT-SN}) \qquad \frac{E[x] \in \text{SN}}{E[x] \in \mathcal{M}} \quad (\text{SAT-N}) \\[10pt] \frac{E[t'] \in \mathcal{M} \quad E[t] \rightarrow_{\text{whd}} E[t'] \quad \text{prt}(t) \subseteq \text{SN}}{E[t] \in \mathcal{M}} \quad (\text{SAT-W}) \end{array}$$

It is easy to see that $\text{SAT} =_{\text{def}} \{\mathcal{M} \mid \mathcal{M} \text{ is saturated}\}$ is closed under intersection. Therefore the triple $\langle \text{SAT}, \subseteq, \cap \rangle$ forms a complete lattice. The next concept will be fundamental for reasoning with saturated sets.

Definition 6. *Given a set of terms M , the set $\text{cl}(M) := \bigcap \{\mathcal{N} \in \text{SAT} \mid M \cap \text{SN} \subseteq \mathcal{N}\}$ is called the saturated closure or SAT-closure of M .*

$\text{cl}(M)$ is the least saturated superset of $M \cap \text{SN}$. Observe that $M \subseteq \text{cl}(M)$ if and only if $M \subseteq \text{SN}$.

4.1 Saturated sets for the implication

The following construction is standard, we recall it here for the sake of self-containment.

Definition 7. We define $\mathcal{M} \Rightarrow \mathcal{N} = \text{cl}(\{r \in \Lambda \mid \forall s \in \mathcal{M}. rs \in \mathcal{N}\})$, so that $\Rightarrow: \text{SAT} \times \text{SAT} \rightarrow \text{SAT}$ is a binary operation on saturated sets.

Proposition 5 (Soundness). Let $\mathcal{M}, \mathcal{N} \in \text{SAT}$.

1. If $S_x(\mathcal{M}, \mathcal{N}) = \{t \mid \forall s \in \mathcal{M}. t[x := s] \in \mathcal{N}\}$ and $t \in S_x(\mathcal{M}, \mathcal{N})$ then $\lambda xt \in \mathcal{M} \Rightarrow \mathcal{N}$.
2. If $r \in \mathcal{M} \Rightarrow \mathcal{N}$ and $s \in \mathcal{M}$ then $rs \in \mathcal{N}$.

Proof. Straightforward. See for example [9]. □

4.2 SAT valued functions for coinductive predicates

The goal of this section is to develop the main technical contribution of our paper, to construct fixed points of SAT-valued functions, which will be needed later for the semantics of coinductive predicates. For the case of inductive predicates we point to our extended version [11]. The methodology is based on the one developed in section 9.4 of [7] for inductive types. These constructions and their soundness properties will play an essential role in the proof of the adequacy theorem for $\text{AF2}^{M\mu\nu}$.

Let us start by fixing a non-empty set M and by defining for all $n \in \mathbb{N}$, the set of SAT-valued n -ary functions $\text{SAT}_n =_{\text{def}} \{F \mid F : M^n \rightarrow \text{SAT}\}$, with $\text{SAT}_0 = \text{SAT}$. The set SAT_n forms a complete lattice $\langle \text{SAT}_n, \subseteq, \cap \rangle$ with the pointwise inherited definitions $F \subseteq G \Leftrightarrow_{\text{def}} \forall \vec{x} \in M^n. F(\vec{x}) \subseteq G(\vec{x})$ and defining for any $\mathcal{F} \subseteq \text{SAT}_n$, the function $\cap \mathcal{F} : M^n \rightarrow \text{SAT}$ as $(\cap \mathcal{F})(\vec{x}) =_{\text{def}} \cap_{F \in \mathcal{F}} F(\vec{x})$. Through this section we fix a higher-order function $\Phi : \text{SAT}_n \rightarrow \text{SAT}_n$, and tuples of functions $\vec{d} = d_1, \dots, d_n, \vec{f} = f_1, \dots, f_n$ with $d_i, f_i : M \rightarrow M$.

Let us begin with the constructions for coinductive predicates. The idea is that given a coinductive predicate $\nu(\Psi)$, where the interpretation of the predicate transformer Ψ is the function $\Phi : \text{SAT}_n \rightarrow \text{SAT}_n$, its interpretation will be defined as the greatest fixed-point $\nu(\Theta^\Xi)$ of the lower monotization of some operator $\Theta : \text{SAT}_n \rightarrow \text{SAT}_n$ associated to the arbitrary function Φ .

Definition 8. We define $\mathcal{E}_\nu : \text{SAT}_n \rightarrow M^n \rightarrow \mathcal{P}(\Lambda)$ by $\mathcal{E}_\nu(F)(\vec{t}) =_{\text{def}} \{r \in \text{SN} \mid \text{out } r \in \Phi(F)(\vec{d}(\vec{t}))\}$ where $F : M^n \rightarrow \text{SAT}$ and $\vec{t} \in M^n$.

Lemma 1. Let $\Theta_E : \text{SAT}_n \rightarrow \text{SAT}_n$ be defined as $\Theta_E(F)(\vec{t}) =_{\text{def}} \text{cl}(\mathcal{E}_\nu(F)(\vec{t}))$. Then, for any $F \in \text{SAT}_n$, $\mathcal{E}_\nu(F) = \Theta_E(F)$

Proof. It suffices to show that for any $\vec{t} \in M^n$, $\mathcal{E}_\nu(F)(\vec{t}) \in \text{SAT}$. See [11]. □

The post-fixed points of Θ_E are characterized as follows:

Lemma 2. $F \subseteq \Theta_E(F) \Leftrightarrow \forall \vec{t} \in M^n \forall r \in F(\vec{t}). \text{out } r \in \Phi(F)(\vec{d}(\vec{t}))$.

Proof. Straightforward. □

We would like to obtain a greatest fixed-point of Θ_E , but as we do not assume that Φ is monotone, we cannot prove either that Θ_E is monotone. Therefore we cannot apply the Knaster-Tarski fixed-point theorem to Θ_E to obtain a greatest fixed-point of Θ_E , which is what we need to interpret coinductive

predicates. However, we can proceed by using an adequate version of its lower monotonization (see definition 1), $\mathcal{E}_V^\subseteq : \text{SAT}_n \rightarrow M^n \rightarrow \mathcal{P}(\Lambda)$ defined by

$$\mathcal{E}_V^\subseteq(F)(\vec{t}) = \bigcap_{F' \in \text{SAT}_n} \{\mathcal{E}_V(F')(\vec{t}) \mid F \subseteq F'\}$$

It is easy to see that \mathcal{E}_V^\subseteq is monotone. Therefore the operator $\Theta_E^\subseteq : \text{SAT}_n \rightarrow \text{SAT}_n$ given by $\Theta_E^\subseteq(F)(\vec{t}) =_{\text{def}} \text{cl}(\mathcal{E}_V^\subseteq(F)(\vec{t}))$ is also monotone and the function $v(\Phi) \in \text{SAT}_n$ defined by $v(\Phi) =_{\text{def}} \text{gfp}(\Theta_E^\subseteq)$ exists due to the completeness of the lattice $\langle \text{SAT}_n, \subseteq, \cap \rangle$.

Proposition 6. $v(\Phi)$ is a post-fixed point of Θ_E .

Proof. By definition, $v(\Phi)$ is a post-fixed point of Θ_E^\subseteq , that is $v(\Phi) \subseteq \Theta_E^\subseteq(v(\Phi))$. Moreover, it is straightforward to show that $\Theta_E^\subseteq(v(\Phi)) \subseteq \Theta_E(v(\Phi))$, which yields $v(\Phi) \subseteq \Theta_E(v(\Phi))$. \square

Next, we define an operator Θ_I useful to prove the soundness of the inference rule for Mendler corecursion.

Definition 9. Given $\Phi : \text{SAT}_n \rightarrow \text{SAT}_n$ and $F \in \text{SAT}_n$ we define $\mathcal{J}_V : \text{SAT}_n \rightarrow M^n \rightarrow \mathcal{P}(\Lambda)$ as follows: if $\vec{s} \in M^n$ and $\vec{s} \neq \vec{f}(\vec{t})$ then $\mathcal{J}_V(F)(\vec{s}) =_{\text{def}} \emptyset$, and

$$\mathcal{J}_V(F)(\vec{f}(\vec{t})) =_{\text{def}} \left\{ \text{MCoRec } s \mid \begin{array}{l} H \in \text{SAT}_n, \quad r \in H(\vec{t}), \\ s \in \bigcap_{G \in \text{SAT}_n} \left((F \preceq G) \Rightarrow (H \preceq_{\vec{f}} G) \Rightarrow H \preceq_{\text{dof}} \Phi(G) \right) \end{array} \right\}$$

where for any $F, G \in \text{SAT}_n$ and \vec{g} a tuple of functions $g_i : M \rightarrow M$ we define the SAT-set $F \preceq_{\vec{g}} G$ as follows: $F \preceq_{\vec{g}} G =_{\text{def}} \bigcap_{\vec{t} \in M^n} F(\vec{t}) \Rightarrow G(\vec{g}(\vec{t}))$, in particular, $F \preceq G =_{\text{def}} \bigcap_{\vec{t} \in M^n} F(\vec{t}) \Rightarrow G(\vec{t})$.

Finally we define the function $\Theta_I : \text{SAT}_n \rightarrow \text{SAT}_n$ as $\Theta_I(F)(\vec{t}) =_{\text{def}} \text{cl}(\mathcal{J}_V(F)(\vec{t}))$.

Lemma 3. For any $F \in \text{SAT}_n$, $\mathcal{J}_V(F) \subseteq \Theta_I(F)$.

Proof. It suffices to show that for any $\vec{s} \in M^n$, $\mathcal{J}_V(F)(\vec{s}) \subseteq \text{SN}$. See [11]. \square

The pre-fixed points of Θ_I are characterized as follows:

Lemma 4. Let $F \in \text{SAT}_n$.

$$\begin{aligned} \Theta_I(F) \subseteq F &\Leftrightarrow \forall \vec{t} \in M^n. \forall H \in \text{SAT}_n. \forall r \in H(\vec{t}). \\ &\forall s \in \bigcap_{G \in \text{SAT}_n} \left((F \preceq G) \Rightarrow (H \preceq_{\vec{f}} G) \Rightarrow H \preceq_{\text{dof}} \Phi(G) \right). \text{MCoRec } s \in F(\vec{f}(\vec{t})) \end{aligned}$$

Proof. Straightforward. \square

To show the soundness of Mendler corecursion we will use the following

Proposition 7. $v(\Phi)$ is a pre-fixed point of Θ_I .

Proof. We will proceed by extended conventional coinduction, as defined in proposition 1.

Let $\mathfrak{J} =_{\text{def}} v(\Phi)$ and $\mathfrak{J}' =_{\text{def}} \mathfrak{J} \cup \Theta_I(\mathfrak{J})$. We have to prove that $\Theta_I(\mathfrak{J}) \subseteq \Theta_E^\subseteq(\mathfrak{J}')$ and for this, it suffices to show that $\mathcal{J}_V(\mathfrak{J})(\vec{s}) \subseteq \mathcal{E}_V^\subseteq(\mathfrak{J}')(\vec{s})$ for all $\vec{s} \in M^n$.

If $\vec{s} \neq \vec{f}(\vec{t})$ then $\mathcal{J}_V(\mathfrak{J})(\vec{s}) = \emptyset \subseteq \mathcal{E}_V^\subseteq(\mathfrak{J}')(\vec{s})$. For the case $\vec{s} = \vec{f}(\vec{t})$ let us take $\text{MCoRec } s \in \mathcal{J}_V(\mathfrak{J})(\vec{f}(\vec{t}))$ with $r \in H(\vec{t})$, $H \in \text{SAT}_n$ and $s \in \bigcap_{G \in \text{SAT}_n} ((\mathfrak{J} \preceq G) \Rightarrow (H \preceq_{\vec{f}} G) \Rightarrow H \preceq_{\text{dof}} \Phi(G))$. According to the definition of $\mathcal{E}_V^\subseteq(\mathfrak{J}')(\vec{f}(\vec{t}))$ we have to prove that $\text{MCoRec } s \in \mathcal{E}_V(\mathfrak{J}')(\vec{f}(\vec{t}))$ for any $\mathfrak{J}'' \in \text{SAT}_n$ such

that $\mathfrak{J}' \subseteq \mathfrak{J}''$. Let us observe that $\text{MCoRecs } r \in \text{SN}$, for $\mathcal{S}_v(\mathfrak{J})(\vec{f}(\vec{t})) \subseteq \text{SN}$. Therefore, we only need to verify that $\text{out}(\text{MCoRecs } r) \in \Phi(\mathfrak{J}')(\vec{d}(\vec{f}(\vec{t})))$. Since $\Phi(\mathfrak{J}')(\vec{d}(\vec{f}(\vec{t}))) \in \text{SAT}$, it suffices to show that $s(\lambda xx)(\text{MCoRecs } r) \in \Phi(\mathfrak{J}')(\vec{d}(\vec{f}(\vec{t})))$.

We know that $s \in (\mathfrak{J} \preceq \mathfrak{J}') \Rightarrow (H \preceq_{\vec{f}} \mathfrak{J}') \Rightarrow (H \preceq_{\vec{d} \circ \vec{f}} \Phi(\mathfrak{J}'))$ and also that $\lambda xx \in \mathfrak{J} \preceq \mathfrak{J}'$, for $\mathfrak{J} \subseteq \mathfrak{J}'$. Hence, by part 2 of proposition 5, $s(\lambda xx) \in H \preceq_{\vec{f}} \mathfrak{J}' \Rightarrow H \preceq_{\vec{d} \circ \vec{f}} \Phi(\mathfrak{J}')$.

Next, we show that $\text{MCoRecs } s \in H \preceq_{\vec{f}} \mathfrak{J}'$. By part 1 of proposition 5 we only need to show that for all $\vec{t} \in M^n$, $\text{MCoRecs } x \in S_x(H(\vec{t}), \mathfrak{J}'(\vec{f}(\vec{t})))$, which happens if and only if for all $e \in H(\vec{t})$, $(\text{MCoRecs } x)[x := e] \in \mathfrak{J}'(\vec{f}(\vec{t}))$. Therefore we assume $e \in H(\vec{t})$ and need to prove that $\text{MCoRecs } e \in \mathfrak{J}'(\vec{f}(\vec{t}))$, but we have $\text{MCoRecs } e \in \mathcal{S}_v(\mathfrak{J})(\vec{f}(\vec{t}))$ and therefore, by lemma 3, $\text{MCoRecs } e \in \Theta_I(\mathfrak{J})(\vec{f}(\vec{t}))$, but as $\Theta_I(\mathfrak{J})(\vec{f}(\vec{t})) \subseteq \mathfrak{J}'(\vec{f}(\vec{t}))$ we have proven that $\text{MCoRecs } s \in H \preceq_{\vec{f}} \mathfrak{J}'$. Using again the second part of proposition 5, we conclude that $s(\lambda xx)(\text{MCoRecs } s) \in H \preceq_{\vec{d} \circ \vec{f}} \Phi(\mathfrak{J}')$. Finally $r \in H(\vec{t})$ implies that $s(\lambda xx)(\text{MCoRecs } r) \in \Phi(\mathfrak{J}')(\vec{d}(\vec{f}(\vec{t})))$. \square

To finish this section we summarize the soundness properties of the (co)inductive constructions on SAT-valued functions.

Proposition 8 (Soundness of the (co)inductive constructions). *Let $\Phi : \text{SAT}_n \rightarrow \text{SAT}_n$, $\vec{c}, \vec{d}, \vec{f}$ be tuples of functions $c_i, d_i, f_i : M \rightarrow M$, $1 \leq i \leq n$, and $\vec{t} \in M^n$. Then*

1. *If $r \in \Phi(\mu(\Phi))(\vec{t})$ then $\text{in } r \in \mu(\Phi)(\vec{c}(\vec{t}))$.*
2. *If $r \in \mu(\Phi)(\vec{t})$, $H \in \text{SAT}_n$ and $s \in \bigcap_{G \in \text{SAT}_n} ((G \preceq \mu(\Phi)) \Rightarrow (G \preceq_{\vec{f}} H) \Rightarrow \Phi(G) \preceq_{\vec{f} \circ \vec{c}} H)$ then $\text{MRecs } r \in H(\vec{f}(\vec{t}))$.*
3. *If $r \in v(\Phi)(\vec{t})$ then $\text{out } r \in \Phi(v(\Phi))(\vec{d}(\vec{t}))$.*
4. *If $r \in H(\vec{t})$, $H \in \text{SAT}_n$, and $s \in \bigcap_{G \in \text{SAT}_n} ((v(\Phi) \preceq G) \Rightarrow (H \preceq_{\vec{f}} G) \Rightarrow H \preceq_{\vec{d} \circ \vec{f}} \Phi(G))$ then $\text{MCoRecs } r \in v(\Phi)(\vec{f}(\vec{t}))$.*

Proof. Part 3 is consequence of proposition 6 and lemma 2. For part 4 we just use proposition 7 and lemma 4. For the inductive cases we refer to [11]. \square

We are now ready to define an intuitionistic semantics for our logic.

5 Semantics for $\text{AF2}^{M\mu v}$

We present here a realizability semantics for $\text{AF2}^{M\mu v}$ where an object-term will be interpreted as an element of a universe set M , a formula as a SAT-set and a predicate as a SAT-valued function in SAT_n .

Definition 10. *A model for a second-order language \mathcal{L} is a pair $\mathfrak{M} = \langle M, \mathcal{J} \rangle$ where M is a non-empty set and \mathcal{J} is an interpretation function for \mathcal{L} such that $\mathcal{J}(f) : M^n \rightarrow M$, for every n -ary function symbol $f \in \mathcal{L}$ and $\mathcal{J}(P) : M^n \rightarrow \text{SAT}$, for every n -ary predicate symbol $P \in \mathcal{L}$.*

From now on we fix a model $\mathfrak{M} = \langle M, \mathcal{J} \rangle$.

Definition 11. *A state or variable assignment is a function $\sigma : \text{Var} \rightarrow M \cup \text{SAT}_n$ such that $\sigma(x) \in M$ and $\sigma(X^{(n)}) \in \text{SAT}_n$. Given $m \in M$ or $G \in \text{SAT}_n$, the modified assignments $\sigma[x/m]$ and $\sigma[X/G]$ are defined as usual.*

Next, we recursively define the interpretation of terms, predicates and formulas.

Definition 12. Given a variable assignment σ , we define the interpretation function \mathcal{I}_σ , such that $\mathcal{I}_\sigma(r) \in M$, $\mathcal{I}_\sigma(\mathcal{P}) \in \text{SAT}_n$ and $\mathcal{I}_\sigma(A) \in \text{SAT}$, as follows:

- Term interpretation
 - $\mathcal{I}_\sigma(x) = \sigma(x)$
 - $\mathcal{I}_\sigma(f(t_1, \dots, t_n)) = \mathcal{I}(f)(\mathcal{I}_\sigma(t_1), \dots, \mathcal{I}_\sigma(t_n))$
- Predicate interpretation:
 - Predicate variables: $\mathcal{I}_\sigma(X) = \sigma(X)$
 - Predicate symbols: $\mathcal{I}_\sigma(P) = \mathcal{I}(P)$
 - Comprehension predicates: if $\mathcal{F} =_{\text{def}} \lambda \vec{x}.A$, we define $\mathcal{I}_\sigma(\mathcal{F}) = G_{\mathcal{F}}$ where $G_{\mathcal{F}} : M^n \rightarrow \text{SAT}$ is given by $G_{\mathcal{F}}(\vec{m}) = \mathcal{I}_{\sigma[\vec{x}/\vec{m}]}(A)$, for all $\vec{m} \in M^n$.
 - Predicate transformers: if $\Phi =_{\text{def}} \lambda X. \mathcal{P}$, where w.l.o.g., $\mathcal{P} =_{\text{def}} \lambda \vec{x}.A$, we define $\mathcal{I}_\sigma(\Phi) : \text{SAT}_n \rightarrow \text{SAT}_n$ by $\mathcal{I}_\sigma(\Phi)(F)(\vec{m}) = \mathcal{I}_{\sigma[X/F, \vec{x}/\vec{m}]}(A)$, for all $\vec{m} \in M^n$.
This way, it can be proved that for any predicate \mathcal{R} , we have $\mathcal{I}_\sigma(\Phi(\mathcal{R})) = \mathcal{I}_\sigma(\Phi)(\mathcal{I}_\sigma(\mathcal{R}))$.
 - (Co)inductive predicates:
 - * $\mathcal{I}_\sigma(\mu(\Phi)) = \mu(\mathcal{I}_\sigma(\Phi))$
 - * $\mathcal{I}_\sigma(\nu(\Phi)) = \nu(\mathcal{I}_\sigma(\Phi))$
 where of course, the operators μ and ν on the right-hand side of the equalities refer to the constructions on SAT-valued functions developed in section 4.
- Formula interpretation:
 - $\mathcal{I}_\sigma(\mathcal{P}(t_1, \dots, t_n)) = \mathcal{I}_\sigma(\mathcal{P})(\mathcal{I}_\sigma(t_1), \dots, \mathcal{I}_\sigma(t_n))$
 - $\mathcal{I}_\sigma(A \rightarrow B) = \mathcal{I}_\sigma(A) \Rightarrow \mathcal{I}_\sigma(B)$
 - $\mathcal{I}_\sigma(\forall x A) = \bigcap \{ \mathcal{I}_{\sigma[x/m]}(A) \mid m \in M \}$
 - $\mathcal{I}_\sigma(\forall X A) = \bigcap \{ \mathcal{I}_{\sigma[X/G]}(A) \mid G \in \text{SAT}_n \}$

We observe that as equations are a special case of a second-order universal formula, there is no need to give a specific semantics for them. However we are only interested in models that satisfy a set of equations in the following sense.

Definition 13. Let $\mathfrak{M} = \langle M, \mathcal{I} \rangle$ be a model and σ be a state. We say that the interpretation \mathcal{I}_σ satisfies the equation $r = s$ if and only if $\mathcal{I}_\sigma(r) = \mathcal{I}_\sigma(s)$. Moreover if \mathbb{E} is a set of equations, we say that \mathcal{I}_σ satisfies \mathbb{E} if and only if \mathcal{I}_σ satisfies every equation in \mathbb{E} .

Now we can prove the main theorem of this paper.

Theorem 1 (Adequacy or soundness). Let $\mathfrak{M} = \langle M, \mathcal{I} \rangle$ be a model such that the interpretation \mathcal{I}_σ satisfies the set of equations \mathbb{E} . If $\Gamma \vdash_{\mathbb{E}} t : A$, with $\Gamma = \{x_1 : A_1, \dots, x_n : A_n\}$ and for all $1 \leq i \leq n$, $r_i \in \mathcal{I}_\sigma(A_i)$ then $t[\vec{x} := \vec{r}] \in \mathcal{I}_\sigma(A)$.

Proof. Induction on $\Gamma \vdash_{\mathbb{E}} t : A$. We discuss the case for the rule (vI), for the remaining rules see [11]. We need to show that $(\text{MCoRec.sr})[\vec{x} := \vec{r}] \in \mathcal{I}_\sigma(\nu(\Phi)(\vec{f}(\vec{r})))$. That is, $(\text{MCoRec.sr})[\vec{x} := \vec{r}] r[\vec{x} := \vec{r}] \in \nu(\mathcal{I}_\sigma(\Phi))(\vec{j})$, where $\vec{j} = \mathcal{I}_\sigma(f_1(t_1)), \dots, \mathcal{I}_\sigma(f_n(t_n))$. The I.H. yields $s[\vec{x} := \vec{r}] \in \mathcal{I}_\sigma(\forall X (\nu(\Phi) \subseteq X \rightarrow \mathcal{H} \subseteq_{\vec{f}} X \rightarrow \mathcal{H} \subseteq_{\vec{d} \circ \vec{f}} \Phi(X)))$. From this and by defining $\Phi' = \mathcal{I}_\sigma(\Phi)$, $\vec{f} = \mathcal{I}_\sigma(f_1), \dots, \mathcal{I}_\sigma(f_n)$, $\vec{d} = \mathcal{I}_\sigma(d_1), \dots, \mathcal{I}_\sigma(d_n)$ and $H = \mathcal{I}_\sigma(\mathcal{H})$ it is easy to verify that $s[\vec{x} := \vec{r}] \in \bigcap_{G \in \text{SAT}_n} ((\nu(\Phi') \preceq G) \Rightarrow (H \preceq_{\vec{f}} G) \Rightarrow (H \preceq_{\vec{d} \circ \vec{f}} \Phi'(G)))$. Moreover we also have $r[\vec{x} := \vec{r}] \in H(\vec{l})$, where $\vec{l} = \mathcal{I}_\sigma(t_1), \dots, \mathcal{I}_\sigma(t_n)$, by I.H. Therefore we can apply part 4 of proposition 8 to conclude that $\text{MCoRec.sr}[\vec{x} := \vec{r}] r[\vec{x} := \vec{r}] \in \nu(\Phi')(\vec{f}(\vec{l}))$, which is equivalent to $(\text{MCoRec.sr})[\vec{x} := \vec{r}] \in \mathcal{I}_\sigma(\nu(\Phi)(\vec{f}(\vec{r})))$. \square

5.1 Strong normalization

The strong normalization property for the logic $\text{AF2}^{M\mu\nu}$ can be proved by adapting the proof of AF2 which embeds this logic into its propositional fragment, system F (see [4]). However, our semantics of saturated sets allows for an easy proof of strong normalization which is a direct consequence of the adequacy theorem. Let us start by building a model and an interpretation that satisfies a given set of equations \mathbb{E} as required by the adequacy theorem.

Definition 14. Given a judgement $\Delta =_{\text{def}} \Gamma \vdash_{\mathbb{E}} t : A$ we define a model $\mathfrak{M}_{\Delta} = \langle M, \mathcal{I} \rangle$ as follows:

- Let $\approx_{\mathbb{E}}$ be the binary relation on terms given by $r = s \Leftrightarrow_{\text{def}} \mathbb{E} \triangleright r = s$. It is easy to prove that $\approx_{\mathbb{E}}$ is an equivalence relation.
- The universe of \mathfrak{M}_{Δ} is the set $M = \text{Term}_{\mathcal{L}} / \approx_{\mathbb{E}}$, of the equivalence classes $[t]$ of the relation $\approx_{\mathbb{E}}$.
- The interpretation function \mathcal{I} is defined as follows:
 - $f^{\mathcal{I}} : M^n \rightarrow M$, $f^{\mathcal{I}}([t_1], \dots, [t_n]) =_{\text{def}} [f(t_1, \dots, t_n)]$
 - $P^{\mathcal{I}} : M^n \rightarrow \text{SAT}$, $P^{\mathcal{I}}([t_1], \dots, [t_n]) =_{\text{def}} \text{cl}(\{s \in \Lambda \mid \Gamma \vdash_{\mathbb{E}} s : P(t_1, \dots, t_n)\})$

It is easy to see that the interpretation function is well-defined and therefore \mathfrak{M}_{Δ} is a model.

The next lemma shows that in \mathfrak{M}_{Δ} term interpretation is given by a specific substitution.

Lemma 5. Let σ be a state and $r \in \text{Term}_{\mathcal{L}}$ such that $\text{Var}(r) = \vec{x}$. If $\sigma(x_i) = [s_i]$ then $\mathcal{I}_{\sigma}(r) = [r[\vec{x} := \vec{s}]]$.

Proof. Induction on r . □

We can now define an interpretation that satisfies a given set of equations \mathbb{E} .

Lemma 6. For any judgement $\Delta =_{\text{def}} \Gamma \vdash_{\mathbb{E}} t : A$ there is a state σ of \mathfrak{M}_{Δ} such that the interpretation \mathcal{I}_{σ} satisfies \mathbb{E} .

Proof. We define the state σ of \mathfrak{M}_{Δ} , as follows:

- For any first-order variable x , $\sigma(x) = [x]$.
- For any second-order variable X , $\sigma(X) = G$, where

$$G([t_1], \dots, [t_n]) =_{\text{def}} \text{cl}(\{s \in \Lambda \mid \Gamma \vdash_{\mathbb{E}} s : X(t_1, \dots, t_n)\}).$$

It is easy to verify that the state is well-defined. Moreover \mathcal{I}_{σ} satisfies \mathbb{E} , for if $r = s \in \mathbb{E}$ then $r \approx_{\mathbb{E}} s$ and therefore $[r] = [s]$. But, if $\text{Var}(r) = \vec{x}$ and $\text{Var}(s) = \vec{y}$, then by definition of σ and by lemma 5 we have $\mathcal{I}_{\sigma}(r) = [r[\vec{x} := \vec{x}]] = [r] = [s] = [s[\vec{y} := \vec{y}]] = \mathcal{I}_{\sigma}(s)$. □

The strong normalization of $\text{AF2}^{M\mu\nu}$ is now easily gained from lemma 6 and the adequacy theorem.

Theorem 2 (Strong normalization of $\text{AF2}^{M\mu\nu}$). If $\Gamma \vdash_{\mathbb{E}} t : A$ then t is strongly normalizing

Proof. Assume Δ is the judgement $\Gamma \vdash_{\mathbb{E}} t : A$, with $\Gamma = \{x_1 : A_1, \dots, x_k : A_k\}$. By lemma 6 the set of equations \mathbb{E} is satisfied by an interpretation \mathcal{I}_{σ} in the model \mathfrak{M}_{Δ} . Moreover, we have $x_i \in \mathcal{I}_{\sigma}(A_i)$, for $\mathcal{I}_{\sigma}(A_i)$ is a SAT-set and every SAT-set contains all variables. Therefore the adequacy theorem yields $t = t[\vec{x} := \vec{x}] \in \mathcal{I}_{\sigma}(A)$. Finally, as $\mathcal{I}_{\sigma}(A) \subseteq \text{SN}$, we get $t \in \text{SN}$ which implies that t strongly normalizes. □

6 Related Work

Nowadays, there are several lines of research concerning fixed-point logics in computer science. In relation to our work we can mention for instance [12] which presents a sequent calculus for positive equi-(co)inductive equational definitions and which handles conventional (co)iteration only. In this paper the equality relation is primitive and corresponds to unification with respect to $\beta\eta$ -reduction. Moreover, the cut-elimination property holds only after restricting the coinductive rules. Recently [2] develops an extension of the linear logic MALL and a focused proof system for it where the mechanism of conventional equi-(co)inductive definitions is similar to ours. In this weak normalizable logic, which only handles (co)iteration, all predicate operators are assumed to be monotone, proofs of functoriality are given for positive definitions and the treatment of equality originates from logic programming. Finally we mention the work of [1] which is closer to ours and presents two strongly normalizing propositional logics (type systems) with Mendler-style positive equi-(co)inductive types whose semantics of so-called guarded saturated sets makes heavy use of transfinite ordinal recursion, which obliges to restrict the (co)iteration rules by means of a kind system that distinguishes between guarded and unguarded types. On the other hand this feature allows for a definition of a system of sized types that encompasses primitive (co)recursion and course of value recursion.

7 Closing remarks

We have presented the logic $\text{AF2}^{M\mu\nu}$, an extension of the second order logic AF2 with Mendler-style primitive (co)recursion over least and greatest fixed points of predicate transformers. To our knowledge, this is the first such extension that includes Mendler-style (co)inductive predicates while keeping the strong normalization property. Thus, the programs extracted from the termination statements of functions are guaranteed to terminate, independently of the syntactical shape of the proof and therefore the particular methodologies to show termination, like the one in [6] are not needed. Based on the concept of monotonization of an operator we have developed a realizability semantics of SAT-sets and SAT-valued functions for (co)inductive predicates that does not employ the usual positivity restriction. This was first achieved in [7] for essentially the propositional inductive fragment of our logic. Furthermore, our adequacy theorem does not require any ordinal recursion in contrast to the work in [14, 15]. The iso-style of our (co)inductive definitions allows to define data types in a similar way to the definition mechanisms of functional programming by using a generic constructor (destructor), a feature that can be easily enhanced to use several specific constructors by means of clausal definitions (see [10]), a mechanism which also allows not to use neither existential nor restricted formulas. By means of examples, we have shown the suitability of the logic to extract programs from proofs. However, the concept of formal data type and other semantical foundations of the program extraction method, like the issue of equality for coinductive data types, as well as the development of more sophisticated case studies, are work in progress.

Acknowledgements

We are thankful to the anonymous referees for the helpful comments regarding the contents of this paper, in particular for the gentle hint to include the conatural numbers as an example. We also gratefully acknowledge Martha Elena Buschbeck Alvarado for improving the English manuscript.

References

- [1] A. Abel (2007): *Mixed Inductive/Coinductive Types and Strong Normalization*, LNCS 4087, Springer, pp. 286-301, doi:10.1007/978-3-540-76637-7_19
- [2] D. Baelde (2011): *Least and Greatest Fixed Points in Linear Logic*. Accepted for publication at the ACM Transactions on Computational Logic, <http://arxiv.org/abs/0910.3383v4>
- [3] J.L. Krivine & M. Parigot (1990): *Programming with Proofs*. In *Journal of Information Processing and Cybernetics EIK (Formerly Elektronische Informationsverarbeitung und Kybernetik)* 26(3), pp. 149-167.
- [4] J.L. Krivine (1993): *Lambda-Calculus, Types and Models*. Ellis Horwood Series in Computers and their Applications. Ellis Horwood, Masson.
- [5] D. Leivant (1983): *Reasoning about Functional Programs and Complexity Classes associated with Type Disciplines..* In *Proceedings of 24th Annual Symposium on Foundations of Computer Science*, IEEE Computer Science Press, pp.460-469, doi:10.1109/SFCS.1983.50
- [6] P. Manoury & M. Simonot (1994): *Automatizing termination proofs of recursively defined functions..* *Theoretical Computer Science* 135, pp. 319-343, doi:10.1016/0304-3975(94)00021-2
- [7] R. Matthes (1999): *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types*. Dissertation Universität München, <http://www.irit.fr/~Ralph.Matthes/dissertation/matthesdiss.pdf>
- [8] N.P. Mendler (1987): *Recursive Types and Type Constraints in Second-Order Lambda Calculus*. In *Proceedings of the 2nd Annual Symposium on Logic in Computer Science*, Ithaca N.Y. IEEE Computer Society Press, pp. 30-36.
- [9] F.E. Miranda-Perea (2009): *Two Extensions of System F with (Co)iteration and Primitive (Co)recursion Principles*. *Theoretical Informatics and Applications* 43(4), pp. 703–766, doi:10.1051/ita/2009015
- [10] F. E. Miranda-Perea (2005): *Realizability for Monotone and Clausular (Co)inductive Definitions*. *Electronic Notes in Theoretical Computer Science* 123, pp. 179-193, doi:10.1016/j.entcs.2004.04.048
- [11] F.E. Miranda-Perea & L. C. González-Huesca (2012): *Mendler-style Iso-(Co)inductive predicates: a strongly normalizing approach (Extended Version)*. Technical report, Facultad de Ciencias UNAM. Available upon request.
- [12] A. Momigliano & A. Tiu (2003): *Induction and Co-induction in Sequent Calculus*. LNCS 3085, Springer, pp. 293-308, doi:10.1007/978-3-540-24849-1_19
- [13] M. Parigot (1989): *On the Representation of Data in Lambda-Calculus*. LNCS 440, doi:10.1007/3-540-52753-2_47
- [14] M. Parigot (1992): *Recursive programming with proofs*. *Theoretical Computer Science* 94, pp. 335-356, doi:10.1016/0304-3975(92)90042-E
- [15] C. Raffalli (1993): *Data Types, Infinity and Equality in System AF2*. LNCS 832, Springer, pp. 280-294, doi:10.1007/BFb0049337
- [16] T. Uustalu (1998): *Natural deduction for intuitionistic least and greatest fixedpoint logics, with an application to program construction* (PhD thesis). Dissertation TRITA-IT AVH 98:03, Dept. of Teleinformatics, Royal Inst of Technology (KTH), Stockholm.
- [17] T. Uustalu & V. Vene (2002): *Least and greatest fixed-points in intuitionistic natural deduction*. *Theoretical Computer Science* 272, pp. 315-339, doi:10.1016/S0304-3975(00)00355-8